Basic Data Structures

Python has a wide variety of built-in types for storing anything from numbers and text (e.g., `int`, `float`, `str`) to common data structures (e.g., `list`, `tuple`, `dict`).

Content Learning Objectives

After completing this activity, students should be able to:

- Reference a specific element of a sequence by an index.
- Compare and contrast numeric and sequence data types.
- Create a dictionary of strings and look up values by key.

Process Skill Goals

During the activity, students should make progress toward:

- Providing feedback on how well other team members are working. (Teamwork)

Facilitation Notes

All three models involve completing tables interactively using a Python Shell. Encourage teams to complete them as quickly as possible, without thinking about individual results. The questions are designed for students to go back and interpret the results as a whole. If they spend too long discussing the table, they won't finish the questions.

The questions on Model 1 are short and may not lead to much whole-class discussion. Have neighboring teams compare answers and discuss any differences that arise. Some students may be aware of negative indexes, but you may want to avoid them at this point. A future activity will explore more advanced topics with indexing and slicing.

On Model 2, students might ask when to use a tuple instead of a list. Avoid long discussions about mutability, and instead focus on everyday examples. Some data are inherently lists: class rosters, grocery lists, etc. Other data are inherently tuples: coordinates, dates, etc. Have at least 2–3 teams report out #16 and encourage different ways of summarizing the main ideas.

When teams get to Model 3, remind the managers to keep track of time. Reserve some time at the end to report out #20 and #23. Discuss the similarities and differences of lists, tuples, and dictionaries. If time permits, have teams brainstorm everyday examples of each data type.

Model 1 Lists

A variable can hold multiple values in the form of a ***list***. The values are separated by commas and wrapped in square brackets. For example:

```
primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

Each ***element*** of the list can be referenced by an ***index***, which is the sequential position starting at 0. For example, `primes[4]` is 11.

index	0	1	2	3	4	5	6	7	8	9
value	2	3	5	7	11	13	17	19	23	29

Do not type anything yet! Read the questions first!

Python code	Shell output
`odd = [1, 3, 5, 7]`	
`odd`	[1, 3, 5, 7]
`odd[2]`	5
`odd[4]`	IndexError
`len(odd)`	4
`number = odd[1]`	
`number`	3
`odd[1] = 2`	
`odd`	[1, 2, 5, 7]
`number`	3

Questions (10 min) **Start time:** ________

1. What is the index of the second element of `primes`? What is the value at that index?

The index is 1. The value is 3.

2. How does the index number compare to the position of the element?

One less — index zero is the first element.

3. Type each line of code in a Python Shell and write the corresponding output in the space above. If an error occurs, write what type of error. Place an asterisk (*) next to any output for which you were surprised.

4. How did you reference the value of the 3rd element of `odd`?

odd[2]

5. What did the output of the len() function tell you about the list?

The length of the list

6. The output of Model 1 displayed an error. Explain the reason for the error.

It says "IndexError: list index out of range". The maximum index is 3 (i.e., length - 1), so 4 is too big.

7. Write a statement that assigns a list of three integers to the variable `run`.

run = [1, 2, 3]

8. Write a statement that assigns the value 100 to the last element of `run`.

run[2] = 100

9. Write a statement that assigns the first value of `run` to a variable named `first`.

first = run[0]

Model 2 Sequences

Lists and strings are examples of ***sequence*** types. Complete the table below to explore how sequences work.

Python code	Shell output
`seq1 = "one two"`	
`type(seq1)`	<class 'str'>
`len(seq1)`	7
`seq1[1]`	'n'
`seq1[1] = '1'`	TypeError: 'str' object does not support item assignment
`seq2 = "one", "two"`	
`type(seq2)`	<class 'tuple'>
`len(seq2)`	2
`seq2[1]`	'two'
`seq2[1] = '1'`	TypeError: 'tuple' object does not support item assignment
`seq3 = ["one", "two"]`	
`type(seq3)`	<class 'list'>
`seq3[1]`	'two'
`seq3[1] = 1`	
`seq4 = ("one", 1)`	
`type(seq4)`	<class 'tuple'>
`number = 12345`	
`number[3]`	TypeError: 'int' object is not subscriptable

Questions (15 min) Start time: ________

10. How does a sequence type differ from a number? (See the last row of the table.)

The value of each element of a sequence is accessible by an index

11. What are the names of the three sequence types introduced in Model 2?

list, string, and tuple

12. How does the syntax of creating a tuple differ from creating a list?

Tuples use parentheses, and lists use square brackets.

13. Is there more than one way (syntax) to create a tuple? Justify your answer.

Yes, the parentheses are optional as long as you use the comma operator. For example, the variable `seq2` is a tuple.

14. Which sequence types allow their elements to be changed? Which do not?

Only lists allow elements to be changed; strings and tuples do not support assignment.

15. Is it possible to store values of different types in a sequence? If yes, give an example from the table; if no, explain why not.

Yes; the `seq4` tuple ("one", 1) stores both a string and an integer.

16. Summarize the difference between lists and tuples. How do they look differently, and how do they work differently?

Tuples use parentheses, but lists use square brackets. Tuples cannot be modified, but lists can be modified.

Model 3 Dictionaries

In Python, a ***dictionary*** stores "`key: value`" pairs. The pairs are separated by commas and wrapped in curly braces. For example:

```
elements = {'C': 'carbon', 'H': 'hydrogen', 'O': 'oxygen', 'N': 'nitrogen'}
```

Key	Value
`'C'`	`'carbon'`
`'H'`	`'hydrogen'`
`'O'`	`'oxygen'`
`'N'`	`'nitrogen'`

In contrast to sequence types, a dictionary is a ***mapping*** type. Values are referenced by ***keys***, rather than by indexes.

Type the `elements` dictionary above into a Python Shell, and then complete the following table to explore how it works.

Python code	Shell output
`type(elements)`	<class 'dict'>
`elements.keys()`	dict_keys(['C', 'H', 'O', 'N'])
`elements.values()`	dict_values(['carbon', 'hydrogen', 'oxygen', 'nitrogen'])
`elements['C']`	'carbon'
`atom = 'N'`	
`elements[atom]`	'nitrogen'
`elements[N]`	NameError: name 'N' is not defined
`elements['nitrogen']`	KeyError: 'nitrogen'
`elements[1]`	KeyError: 1
`len(elements)`	4
`elements['B'] = 'Boron'`	
`elements.items()`	dict_items([('C', 'carbon'), ('H', 'hydrogen'), ...])

Questions (20 min)

Start time: ________

17. List all the keys stored in the `elements` dictionary after completing the table.

The keys are `'C'`, `'H'`, `'N'`, `'B'`, and `'O'`.

18. What is the data type of the keys in the `elements` dictionary?

The keys are all strings. (Note: there is no "char" type in Python.)

19. Explain the reason for the error after entering each of the following lines:

a) `elements[N]` The letter N is treated as a variable name, and it's undefined.

b) `elements['nitrogen']` The string `"nitrogen"` is not one of the keys.

c) `elements[1]` The integer 1 is also not a key, and there are no indexes.

20. Ignoring the "dict_items()" part, describe the contents and type of data returned by the `items()` method.

It appears to return a list of tuples (of key-value pairs). In reality, the function returns a "view" of the dictionary. See https://docs.python.org/3/library/stdtypes.html#dict-views.

21. Write a Python expression that creates a dictionary for the seven days of the week, i.e., Sun=1, Mon=2, Tue=3, etc. Assign the dictionary to the variable `dow`.

```
dow = {'Sun': 1, 'Mon': 2, 'Tue': 3, 'Wed': 4, 'Thu': 5, 'Fri': 6, 'Sat': 7}
```

22. If you assign two different values to the same key (i.e., two assignment statements with one value each), which value is stored in the dictionary? Justify your answer with an example.

If you were to assign `dow['Sun'] = 8` followed by `dow['Sun'] = 0`, then 0 would replace the previous value.

23. Another way to store the data in Model 3 is to use two lists:

```
keys = ['C', 'H', 'O', 'N']
vals = ['carbon', 'hydrogen', 'oxygen', 'nitrogen']
```

What is a disadvantage of this approach? Explain your reasoning.

It's more difficult to insert new items: you have to write two assignment statements instead of one. It's even more difficult to update items: you have to determine the index of the key and replace the corresponding value in the other list.